Minimizing the cardinality of a real-time task set by automated task clustering

Antoine Bertout, Julien Forget, Richard Olejnik

Minimizing the cardinality of a real-time task set by automated task clustering

Antoine Bertout, Julien Forget and Richard Olejnik
Laboratoire d'Informatique Fondamentale de Lille
Université Lille 1, France
{antoine.bertout,julien.forget,richard.olejnik}@lifl.fr

ABSTRACT

The objective of this paper is first to properly define the notion of task clustering. This is the process of automatically mapping functionalities (blocks of code corresponding to a high-level feature) with real-time constraints to tasks (or threads). We aim at reducing the number of tasks functionalities are mapped to, while preserving the schedulability of the initial system. Second, our goal is to expose the complexity of the problem and to sketch methods we will propose for solving this problem. We consider independent tasks running on a single processor.

1. INTRODUCTION

Our work falls within the scope of real-time systems programming. Usually, real-time system developers design a system as a set of functionalities with real-time constraints. A functionality is here considered a block of code corresponding to a high-level feature. Implementing such systems requires to map each functionality to a real-time task (thread). On the one hand, the number of those functionalities is quite high. For instance, it ranges from 500 to 1000 in the flight control system of an aircraft or of a space vehicle [6, 10]. On the other hand, a large number of threads implies, a significant time overhead in context switching [23, 13] and an important memory footprint (e.g. task control block, size of the stack, etc.). Thus, the number of tasks supported by embedded real-time operating systems is limited, rarely over one hundred and developers cannot map each functionality to a different task. This mapping is currently mainly performed manually and, given the number of functionalities to process, this work can be tedious and error-prone.

In our work, we address this question from the scheduling point of view. We model a system as a set of tasks with real-time constraints, where each task is characterized by an execution time, an activation period and a deadline, in the same way as Liu and Layland's task model [16]. With respect to this model, functionalities can simply be considered as finer grain tasks, while threads are just coarser tasks. Thus, mapping functionalities to tasks amounts to gathering several tasks into a single one, which we call *task clustering*. Clustering several tasks implies to choose only one deadline for the cluster, which effectively reduces some task deadlines. As a consequence, we have to check that the system schedulability is preserved after the clustering. Our objective is to automate the clustering, so as to reach a minimal task number, while preserving the system schedulability.

Related Work.

In the literature, task clustering is most often studied in the context of distributed systems implementation, where it consists in distributing a set of tasks over a set of computing nodes (processors or cores). This is different from our context, because in the distributed systems context a cluster corresponds to the set of tasks allocated to the same computing resource. For instance, [20, 1] aim at minimizing communications by clustering tasks that communicate a lot. The approaches in [19, 11] cluster tasks based on communications, in order to reduce the system makespan. The number of tasks of the resulting implementation is however not reduced.

Functionality to task mapping is known as runnable-to-task mapping and is identified as a step of the development process in the augmented real-time specification for AUTomotive Open System ARchitecture (AUTOSAR) [5]. This document and [23] also provide guidelines defining under which conditions runnables can be mapped to the same tasks. Authors in [26] propose an automated mapping in that context, but that work is restricted to functionalities that have deadlines equal to their periods. In [7, 18], the authors study the multi-task implementation of multi-periodic synchronous programs and must allocate the different elements of the program to tasks. The clustering is out of the scope of [18], while the heuristic proposed in [7] is very specific to the language structure.

In [22], authors aim at reducing the number of tasks in order to reduce the complexity of the scheduling problem. However, they only focus on functional requirements to group tasks, without considering timing constraints.

This research.

The number of possible clusterings of a task set is equal to the number of partitions of the set, which is close to the *Bell number* [21]. The Bell number is exponential with respect to the cardinality of the set. Thus, given the huge number of possibilities to explore, we motivate the use of a heuristic to tackle the task clustering problem. We also study the schedulability tests that can be applied to first, check the schedulability of a clustering and second, to constitute a relevant heuristic cost function. For now, we do not consider communications and the execution platform is made up of a single processor. These are strong restrictions, which will be lifted in future work. The aim of the present paper is to properly define the problem and to study it in a simple setting, so as to serve as a basis for future work.

Organization.

The rest of the paper is organized as follows. In Section 2, we describe our clustering model. Section 3 is dedicated to the complexity of the task clustering problem. We adress the question of schedulability in Section 4. We describe the current status and the future work involved in Section 5.

2. PROBLEM DEFINITION

Our model, illustrated in Figure 1, is based on Liu and Layland's model [16]. A system consists of a synchronous (i.e. with offsets equal to zero) set of real-time tasks $\mathcal{S} = (\{\tau_i(C_i, D_i, T_i)\}_{1\le i\le n})$ where C_i is the worst-case execution time (WCET) of τ_i, T_i is the activation period, D_i is the relative deadline with $D_i \le T_i$. We denote $\tau_{i.k}$ the $(k+1)^{th}$ $(k \ge 0)$ instance, or *job*, of τ_i. The job $\tau_{i.k}$ is released at time $o_{i.k} = kT_i$. Every job $\tau_{i.k}$ must be completed before its absolute deadline $d_{i.k} = o_{i.k} + D_i$

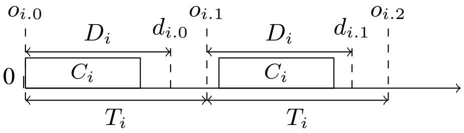


Figure 1: Task Diagram.

2.1 Scheduling

In this paper, we focus on priority-based scheduling policies, either fixed-job with Earliest Deadline First [16](EDF) or fixed-task priority policies with Deadline Monotonic [14](DM).

Let $\mathcal{J}$ denote an infinite set of job, i.e., $\mathcal{J} = \{\tau_{i.k}, 1 \le i \le n, k \in \mathbb{N}\}$. Given a priority assignment Φ where 0 is the lowest priority, we define two functions $s_\Phi, e_\Phi : \mathcal{J} \to \mathbb{N}$, where $s_\Phi(\tau_{i.k})$ is the start time and $e_\Phi(\tau_{i.k})$ is the completion time of $\tau_{i.k}$ in the schedule produced by Φ.

DEFINITION 1. *Let $\mathcal{S} = (\{\tau_i\}_{1\le i\le n})$ be a task set and Φ be a priority assignment. S is* schedulable *under Φ if and only if:* $\forall \tau_{i.k}, e_\Phi(\tau_{i.k}) \le d_{i.k} \wedge s_\Phi(\tau_{i.k}) \ge o_{i.k}$

In the sequel, we will also rely on the notion of *laxity*.

DEFINITION 2. *Laxity L (or* slack time*) indicates the maximum delay that can be taken by the task without exceeding its deadline: $L_i = D_i - C_i$.*

2.2 Clustering

Clustering τ_i and τ_j, where $D_i \le D_j$, produces a cluster τ_{ij} with the following parameters:

$C_{ij} = C_i + C_j$

$T_{ij} = T_i = T_j$

$D_{ij} = D_i$

The cluster deadline is the shortest of the two tasks. Taking the minimum deadline ensures we respect both initial deadlines, even though the constraints will be, in general, more stringent than the initial constraints.

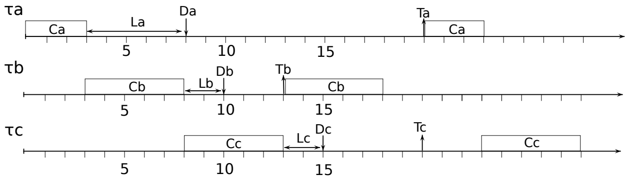
DEFINITION 3. *Let $\mathcal{S} = (\{\tau_i\}_{1\le i\le n})$ be a task set and τ_x and τ_y be two tasks of $\mathcal{S}$ such that $D_x \le D_y$. We say that τ_{xy} is a* valid cluster *if and only if:*

1. $T_x = T_y$
2. $L_x \ge C_y$
3. *The task set obtained after clustering is schedulable*

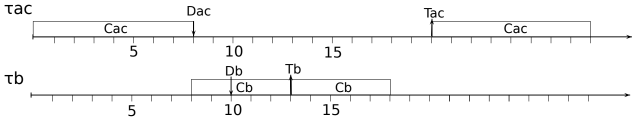
In industrial practices, functionalities of different periods are sometimes mapped together, especially when these functionalities interact a lot, to minimize communication as explained in [24]. This possibility makes the clustering more complex because it requires to manage scheduling inside a cluster. For this reason, we do not deal with this option in this paper. Nevertheless, we could relax this assumption via, e.g., hierarchical scheduling [15].

The laxity test is just an optimization. It is redundant with the schedulability test but it is simpler to check (constant time). Laxity is depicted in Subfigure 2(a).

A schedulable system might become non schedulable after clustering, as illustrated in Figure 2. Indeed, we notice in Subfigure 2(b) that the task τ_b misses its first deadline after the clustering of tasks τ_a and τ_c. Thus, we must check the resulting task set schedulability after clustering.



(a) Initial schedulable system of tasks τ_a,τ_b and τ_c under DM.



(b) Resulting unschedulable system after clustering of tasks τ_a and τ_c.

Figure 2: Influence of task clustering on system schedulability.

3. TASK CLUSTERING COMPLEXITY

We aim in this section at emphasizing the complexity of task clustering, which is related to the search space and to the schedulability test applied.

3.1 Search space

Our problem consists in finding a partition of the task set that is schedulable and with a minimum number of subsets. A partition of a set $\mathcal{X}$ is a set of nonempty subsets of $\mathcal{X}$ such that every element n in $\mathcal{X}$ is in exactly one of these subsets. The number of partitions of a set is the Bell number [21]. The Bell number is exponential with respect to the size of $\mathcal{X}$ and can be computed by the following recurrence relation:

$B_{n+1} = \sum\limits_{k=0}^{n} \binom{n}{k} B_k$ with $B_0 = 1$

To give a better idea of the size of the search, notice that for instance, $B_{500} \simeq 10^{844}$.

To be more precise, as we only cluster tasks with identical periods, the search space can be restricted to $\prod\limits_{i=0}^{m} B_{n_i}$ where B_{n_i} is the Bell number of the set of tasks with period T_i and m is the number of different periods of the whole task set. Nevertheless, this number remains exponential.

A naive approach might be to conduct an exhaustive search among all partitions of the initial task set, e.g. by applying partitions generation algorithms [2, 17], checking schedulability for each partition generated and choosing the partition with the least subsets. Nonetheless, our first experimentations show that, even using simple, non exact linear schedulability tests (presented below), this solution is not achievable due to the exponential number of partitions to explore. For instance, experiments conducted on a 2.3GHz Intel Core i7 quad-core with 4GByte memory, from an initial set of 20 tasks, lead to more than several days of computation. Thus, we propose to limit the exploration, by applying a heuristic.

3.2 Heuristic function

We start from an initial task set where each task is considered a cluster with one element, we gradually try to group more and more clusters together to minimize the cardinality of the task set. At each step, we try to group one cluster with another and we have, among the candidates that fullfilled conditions 1,2 and 3, some more or less good possibilities. This could be illustrated by Figure 3 for example. Then, we must select the best candidate. This can be achieved by a heuristic cost (or evaluation) function that estimates which candidate will the most likely lead to the best clustering. We propose to achieve task clustering using classic heuristics based on cost functions, such as greedy Best-first search (greedy BFS), A* algorithm or simulating annealing (SA).

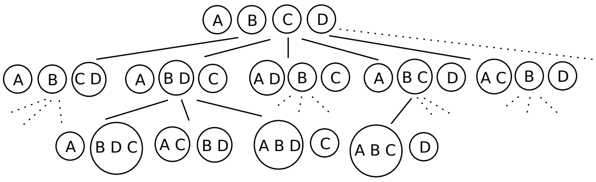


Figure 3: Possible ways to cluster tasks

4. SCHEDULABILITY ANALYSIS

While conditions 1 and 2 adressed earlier can be checked trivially in constant time, condition 3 is more complex. We need a schedulability test to determine a valid task clustering because grouping tasks makes the resulting task set more and more difficult to schedule. Moreover, we need a relevant heuristic cost function to determine the best candidate for the clustering. We want a schedulability test that exhibits some features that might allow us to compare the potential of two task sets. Therefore, in this section, we consider schedulability tests that can be also considered heuristic cost functions.

We present schedulability tests that can be used for clustering under DM and EDF scheduling policies and we detail their ability to be considered a relevant cost function.

A schedulability test is called sufficient if all task sets considered schedulable by the test are actually schedulable. In the same manner, a schedulability test is called necessary if all task sets considered unschedulable by the test are in fact unschedulable. Schedulability tests that are both sufficient and necessary are referred to as exact.

We only consider exact and sufficient tests, thus insuring that the task sets obtained after clustering are schedulable. Indeed, applying sufficient tests means that we might not get the minimum number of clusters but we are sure to still obtain a valid clustering.

4.1 Exact schedulability tests

[8] distinguishes two types of tests: *boolean schedulability tests* and *response time tests*. On the one hand, boolean tests give a boolean answer, determining only whether a task set is schedulable or not, for instance with processor demand analysis (PDA). Thus, they do not exhibit any clear feature that could be considered a heuristic cost function and are not appropriate for our purpose. On the other hand, exact tests based on response time analysis (RTA) provide worst response time for each task and are more suited to be used as cost functions. Indeed, considering a task τ_k with its worst response time denoted R_k, the closer to 1 $\frac{R_k}{D_k}$ is, the less we have margin to group the task τ_k with another. Thus, the sum of each task response time divided by its respective deadline can be used as heuristic cost function. Then, we have a heuristic cost function $h(\mathcal{S})$, such that

$$h(\mathcal{S}) = \sum_{k=0}^{|\mathcal{S}|} \frac{R_k}{D_k}$$

Deadline Monotonic.

RTA [12, 3] of a task τ_i is based on the concept of level-i *busy period*. The level-i busy period is the maximum continuous time interval during which a processor executes tasks of higher or equal priority to the priority of the considered task τ_i, until τ_i finishes its active job. Then, the computation of the worst response time for each task τ_i is based on the length of level-i busy period. RTA for DM can be performed with a pseudo-polynomial time algorithm.

Earliest Deadline First.

Contrary to fixed-task priority (FP) systems, the worst response time is not necessarily found on the first processor busy period in a task set scheduled by EDF [25]. Thus, computing RTA for EDF is more complex and has an exponential complexity.

Even though the RTA for FP has a pseudo-polynomial complexity, early experiments show that the test is quite efficient. The RTA for EDF has an exponential complexity and early experiments seem to show that the test is not practicable (it takes more than several days of computation for 20 tasks).

4.2 Sufficient schedulability conditions

In order to reduce the complexity of the computations, we also considered linear sufficient schedulability tests. Audsley [4] and Devi [9] propose sufficient but not necessary schedulability tests, respectively for DM and EDF in $\mathcal{O}(n)$ complexity. As far as we know, there are no more efficient tests for DM and EDF in linear complexity. The first results show that the test for DM behaves well for clustering and better than that of EDF. Nevertheless, computations with linear test under DM are only 2 times faster than computations with exact RTA test under DM.

Those two sufficient tests actually provide an approximate worst response time for each task. They have a similar form to the exact tests based on RTA. Accordingly, they are also adapted to be used as heuristic cost function.

5. CURRENT STATUS AND FUTURE WORK INVOLVED

We emphasized in this paper that task clustering can not be efficiently achieved by an optimal and exhaustive search but through a heuristic, because of the exponential number of partitions to assess as mentioned in Section 3.1. We explored in this sense the use of sufficient tests and exact tests as heuristic cost functions for DM and EDF.

We are currently working on a heuristic that makes the task clustering feasible. Our preliminary results show that clustering can lead to drastically reducing the number of task, especially when realistic parameters (e.g. with deadlines close to the periods) are used at random task set generation. For instance, we are able to cluster 400 tasks to several dozen in a reasonable time (less than an hour on the machine's configuration cited above) under DM. Results under EDF are less encouraging with high processor utilization factors, probably due to the pessimism of the sufficient test with such settings.

We studied the problem of automatically reducing a large set of independent tasks to a smaller set, while preserving the schedulability of the task set. The current assumption that tasks are independent is quite restrictive and will be lifted in future work. Situations where tasks of different periods may be gathered will also be studied.

6. REFERENCES

[1] A. Ahmadinia, C. Bobda, and J. Teich. Temporal task clustering for online placement on reconfigurable hardware. In *Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on*, pages 359 – 362, Dec. 2003.

[2] J. Arndt. *Matters Computational: Ideas, Algorithms, Source Code*. Springer, 2010.

[3] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[4] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. *Deadline monotonic scheduling*. Citeseer, 1990.

[5] AUTOSAR. *RTE Standard Specifications*.

[6] F. Boniol, P.-E. Hladik, C. Pagetti, F. Aspro, and V. Jégu. A framework for distributing real-time functions. In *Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems*, FORMATS '08, pages 155–169. Springer-Verlag, 2008.

[7] A. Curic. *Implementing Lustre Programs on Distributed Platforms with Real-time Constrains*. PhD thesis, University Joseph Fourier, Grenoble, 2005.

[8] R. I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *Computers, IEEE Transactions on*, 57(9):1261–1276, 2008.

[9] U. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, pages 23 – 30, july 2003.

[10] J. Forget. *A Synchronous Language for Critical Embedded Systems with Multiple Real-Time Constraints*. PhD thesis, Université de Toulouse, 2009.

[11] L. Guodong, C. Daoxu, W. Daming, and Z. Defu. Task clustering and scheduling to multiprocessors with duplication. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, page 8 pp., Apr. 2003.

[12] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[13] E. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.

[14] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250, 1982.

[15] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2):257–269, 2005.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[17] M. Orlov. Efficient generation of set partitions. *Engineering and Computer Sciences, University of Ulm, Tech. Rep*, 2002.

[18] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3):307–338, 2011.

[19] M. Palis, J.-C. Liou, and D. Wei. Task clustering and scheduling for distributed memory parallel architectures. *Parallel and Distributed Systems, IEEE Transactions on*, 7(1):46 –55, Jan. 1996.

[20] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Trans. Parallel Distrib. Syst.*, 6(4):412–420, Apr. 1995.

[21] G.-C. Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5):498–504, 1964.

[22] L. Santinelli, W. Puffitsch, C. Pagetti, and F. Boniol. Scheduling with functional and non-functional requirements: the sub-functional approach. *Work-in-Progress Session of ECRTS 2013*, 2:9, 2013.

[23] O. Scheickl and M. Rudorfer. Automotive real time development using a timing-augmented AUTOSAR specification. *Proceedings of ERTS2008*, 4, 2008.

[24] S. Schliecker, J. Rox, M. Negrean, K. Richter, M. Jersak, and R. Ernst. System level performance analysis for real-time automotive multicore and network architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):979 –992, July 2009.

[25] M. Spuri. Analysis of Deadline Scheduled Real-Time Systems. Research report RR-2772, INRIA, 1996. REFLECS Project.

[26] M. Zhang and Z. Gu. Optimization issues in mapping AUTOSAR components to distributed multithreaded implementations. In *2011 22nd IEEE International Symposium on Rapid System Prototyping (RSP)*, pages 23 –29, May 2011.